ClearSpeed

CSXL User Guide

06-UG-1357 1.13.1.7

30 October 2006

ClearSpeed Technology, Inc.
3031 Tisch Way, Suite 200
San Jose, CA 95128

Tel: 408-557-2067
Fax: 408-557-9054

Email: info@clearspeed.com
Web: www.clearspeed.com

ClearSpeed Technology plc
3110 Great Western Court
Hunts Ground Road
Bristol BS34 8HP
United Kingdom

Tel: +44 (0)117 317 2000
Fax: +44 (0)117 317 2002

Conventions

Convention	Description
`commands`	This typeface means that the command must be entered exactly as shown in the text and the `[Return]` or `[Enter]` key pressed.
`Screen displays`	This typeface represents information as it appears on the screen.
`[Key]` names	Key names appear in the text written with brackets. For example `[Return]` or `[F7]`. If it is necessary to press more than one simultaneously, the key names are linked with a plus (+) sign: Press `[Ctrl]`+`[Alt]`+`[Del]`
Bold-face text	Signal names, instructions and register names are displayed in bold. Selections made via the menu hierarchy of a software application.
Words in *italicized* type	Italics emphasize a point, concept or denote new terms.
ℹ	This symbol indicates important information or instructions.

Contents

1 Introducing CSXL

This guide is intended for anyone who is interested to know how to use CSXL with their application. It is of particular interest to anyone who needs to adapt CSXL to their needs. It is assumed that you are already familiar with the application that you use.

The ClearSpeed accelerated math library (CSXL) is the software component accompanying ClearSpeed's *Advance*™ accelerator board providing optimized implementation of key computational routines. CSXL intercepts calls to the host BLAS library and, if the problem size is suitably large, offloads the processing to the *Advance* accelerator board.

CSXL provides accelerated math functions for proprietary applications as well as many third party applications such as Mathematica and MATLAB, and it is available for a wide range of platforms.

1.1 Improving performance

Performance of the CSXL library may be improved by switching off hyperthreading on Intel processors. This can be typically done via the BIOS. This is a quote from the Release Notes for *Intel Optimized LINPACK Benchmark 2.1.2 for Linux* :

> Intel Optimized Linpack Benchmark is threaded to effectively use multiple processors. Therefore, in MP systems, best performance will be obtained with hyperthreading turned off. This insures that the operating system assigns threads to physical processors only.

1.2 Overview of CSXL integration

CSXL manages the distribution of all math functions over the aggregated resources of the ClearSpeed coprocessors and the host (for example, Xeons, Opterons or Itaniums). A set of environment variables are used to configure the integration between the host and CSXL. This is explained in 2.4, *Configuration variables*, on page 9.

To briefly describe the integration of CSXL, it is worth illustrating the difference between an application with and without CSXL describing the additional components. Figure 1.1 shows a typical application without CSXL.

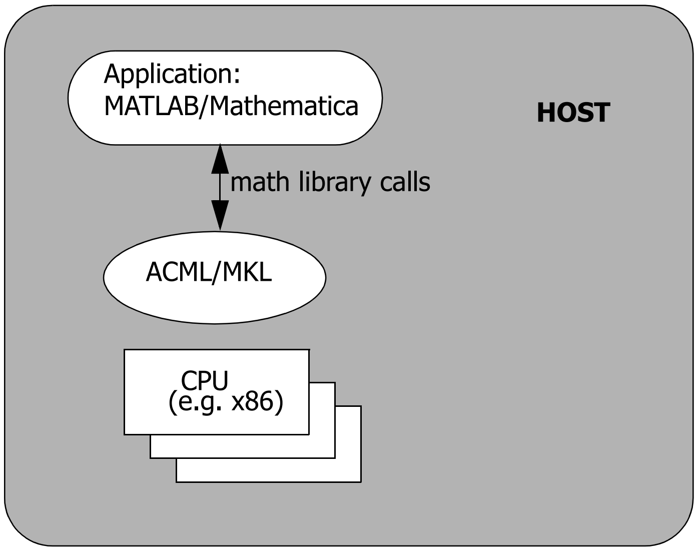

Figure 1.1 Application without CSXL

Figure 1.2 show a typical application with CSXL. There are two component libraries in the CSXL package. The first is the CSXL library. This provides the infrastructure and algorithms to decide which BLAS routines involve CSX600 acceleration and use of the host possibly via n-way BLAS. The second is n-way BLAS. This is a thin software layer using pthreads to provide explicit, dynamic and lightweight control over the number of threads on the host on a call-by-call basis. Then, for example, when the CSX600 processors are selected for inclusion in a BLAS computation, a different number of host cores may be used when the CSX600 processors are not.

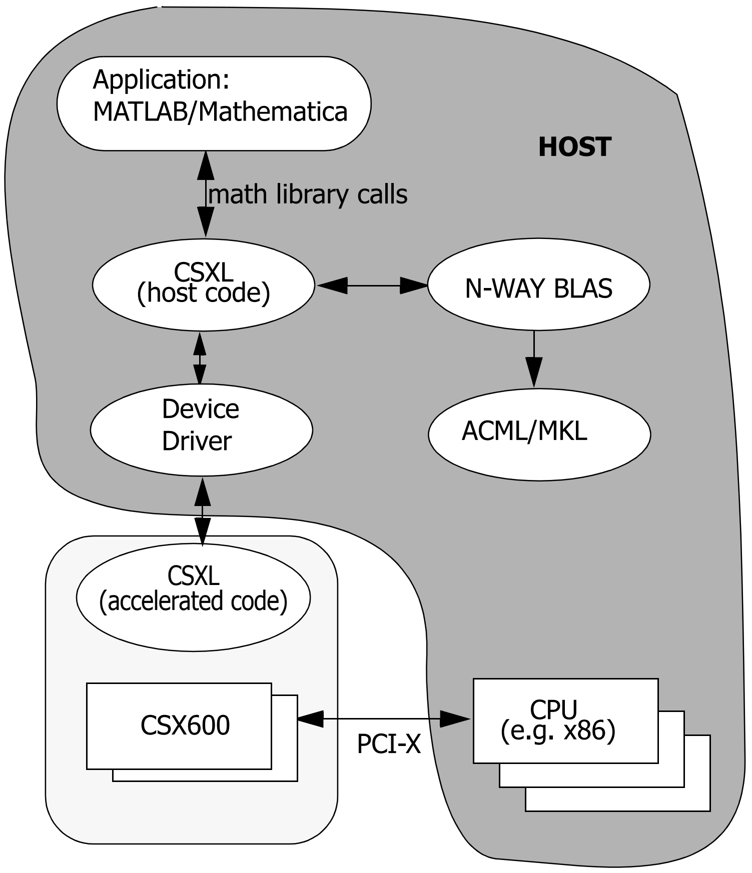

Figure 1.2 Application with CSXL

In Figure 1.2, the CSXL library performs the decision and control activity to distribute work to the appropriate processors. The decision on where data is to be processed is based on factors, such as, the problem size. It can also be controlled by the environment variables (see 2.4, *Configuration variables*, on page 9). As a collective, the CSX600 and x86 (or other) processors are used to provide the best available performance for the math library.

1.3 Hardware and software prerequisites

Besides a ClearSpeed *Advance* accelerator board and software (driver, runtime libraries and CSXL), no additional software or equipment is required over the natural requirements of the application running natively on the host machine. That is to say, you may take the native version of your application and plug in CSXL with its acceleration. The single technical requirement is that the application is linked to a math library using dynamic, or shared, libraries also known as "DLL"s or ".so" files.

1.3.1 Supported host operating systems

The applications and operating systems supported by the CSXL library are shown in Table 1.

	Operating system				
Application	RHEL 3 (IA32)	RHEL 4 (IA32)	RHEL 4 (AMD64/EM64T)	SLES 9.3 (AMD64/EM64T)	RHEL 4 (IA64)
MATLAB R14 SP2	✓	✓[a]	✓[a]	✓[a]	✗
MATLAB R2006a	✗	✓	✓	✓[a]	✗
Mathematica version 5.2	✓	✓	✓	✓	✗

Table 1 CSXL compatibility

a. Limited compatibility: contact ClearSpeed customer support for details.

1.3.2 Supported host BLAS libraries

The following host BLAS libraries are supported:

- ACML
- GOTO-BLAS
- MKL
- ATLAS

1.3.3 Supported compilers

The following compilers are supported:

- For Linux: GNU compiler collection and compatibles
- For Windows XP: Visual C++ 6.0, Visual C++ 2005

2 Installing CSXL

For installation instructions for use with MATLAB and Mathematica on Linux platforms see Chapter 3, *Using CSXL with MATLAB* and Chapter 4, *Using CSXL with Mathematica 5.2*.

2.1 Before you install

- Please note that this is Beta software. Before you install, please check the release notes for any known restrictions.
- This software requires the most recent version of the firmware on the board. Before installing CSXL you must have a compatible version of the drivers installed. See the driver installation instructions for details.

Note: To install the software for your *Advance* accelerator board, you require root access (on Linux) or you need to be in the administrator group (on Windows). Some parts of the installation may have to be carried out by the system administrator.

2.2 Installing the CSXL library on Linux

To install the CSXL library:

1. Login to the Linux machine as root.

   Note: if you use `su` to log in as root, please ensure you use the command `su –` to ensure that the correct environment and path is set up for the following steps.
2. Change to the directory containing the CSXL package.

   This will be the directory where you downloaded the RPM file or the appropriate directory on the CD-ROM. For example:
   ```
   cd ~/clearspeed
   ```
   or:
   ```
   cd /mnt/cdrom/rhe3/csxl
   ```
3. Install the CSXL package. The CSXL package is provided as an RPM file and is installed using the `rpm` command:
   ```
   rpm -i csx600_m512_le-csxl-<version>.i386.rpm
   ```
4. Use the shell script provided to set up the environment variables:
   ```
   source /opt/clearspeed/csx600_m512_le/bin/bashrc
   ```
5. Reset the board:
   ```
   csreset -Av
   ```

2.3 Uninstalling on Linux

The package can be uninstalled using the `rpm -e` command. Packages must be uninstalled in the reverse of the order in which they were installed. You can get a list of the installed ClearSpeed RPMs with the command:

```
rpm -qa | grep csx
```

Then remove the CSXL package using the `rpm` command, for example:

```
rpm -e csx600_m512_le-csxl-<version>
```

2.4 Configuration variables

The following variables are used to configure the integration between the host and CSXL.

2.4.1 General

The behavior of the CSXL library can be controlled by a number of configuration variables. They can either be set in a configuration file or as environment variables. A value specified as an environment variable overrides the value in the configuration file.

CSXL_CONFIG_FILE

This variable specifies the name of the CSXL configuration file. The default value is empty, which means there is no configuration file.

```
CSXL_CONFIG_FILE=/home/jsp/csxl.cfg
```

CS_BLAS_TRACING

Set this variable to `1` to enable profiling in CSXL. By enabling profiling you can:

- Track BLAS calls.
- Check if a function call is running on the host or on the *Advance* accelerator board. `_off_` appears in the trace line if the function call is running on the board.
- Check the logged parameters for traced library functions, such as, the length of vectors or the number of rows and columns of a matrix.
- Check the timing. The final trace parameter `_t` is the time spent in the function in microseconds.

For example:

```
[TRACE dgemm_ transa=N transb=N m=10 n=10 k=10 lda=10 ldb=10 ldc=10 _t=52]
```

The default is 0 (disabled).

```
CS_BLAS_TRACING=1
```

CS_BLAS_TRACE_FILE

When profiling is enabled (by `CS_BLAS_TRACING`), use this variable to specify the name of the output file. On Linux platforms `'tty'` traces to the current terminal. The default value is empty.

```
CS_BLAS_TRACE_FILE=/home/fred/trace.out
```

CS_BLAS_CONSOLE_TRACING

If set to 0, the file name specified by `CS_BLAS_TRACE_FILE` will have the machine name and process/thread ID appended to it. This may be useful when tracing multithreaded or multiprocessor calls to the library. If set to 1, the file name will be used unchanged. If tracing to the console (`CS_BLAS_TRACE_FILE = 'tty'`), this variable *must* be set to 1. The default value is 0.

```
CS_BLAS_CONSOLE_TRACING=0
```

CS_HOST_BLAS

Use this variable to specify which library to use if the *Advance* accelerator board does not accelerate a called function. This is a list of library files (like `LD_LIBRARY_PATH`) so you can specify a number of BLAS libraries which between them implement the full set of BLAS/LAPACK functions. For *both* Linux and Windows, the items in the list are separated by colons (‘`:`’).

Symbols are resolved in lexical order from the list of libraries. In the following example, the `dgemm_` function is resolved in the n-way BLAS library, but `lsame_` is resolved in ACML. The default value is empty. If this variable is not defined, you get an error message. For information on which BLAS libraries are supported, see 1.3.2, *Supported host BLAS libraries*, on page 7.

```
CS_HOST_BLAS=/var/tmp/clearspeed/libnWayBLAS.so:/var/tmp/clearspeed/libacml.so
```

CS_HOST_THREADS

Set this variable to specify the number of processors or threads managing the transfer and pre or postprocessing of data to and from the *Advance* accelerator board. The default is 1. In general, this value should not be changed.

```
CS_HOST_THREADS=1
```

CS_BLAS_HOST_ASSIST_PERCENTAGE

Set this variable to specify how work will be shared between the host processors and the *Advance* accelerator board. For example, if you set it to 20, then 20% of a BLAS call will be performed on the host and 80% on the *Advance* accelerator board. The default is 0 (this means the host will not be used at all). See also 6.2, *Calculating host fraction on Linux*, on page 23.

```
CS_BLAS_HOST_ASSIST_PERCENTAGE=20
```

CS_NUMBER_OF_ASSIST_THREADS

This variable specifies the number of host-side threads used to compute BLAS in conjunction with the *Advance* accelerator board. If the value of `CS_BLAS_HOST_ASSIST_PERCENTAGE` is greater than zero, the value of `CS_NUMBER_OF_ASSIST_THREADS` must be at least 1 and is typically set to one less than the number of host cores. If `CS_BLAS_HOST_ASSIST_PERCENTAGE` is not set or is zero, the `CS_NUMBER_OF_ASSIST_THREADS` is ignored. The default value is 1.

```
CS_NUMBER_OF_ASSIST_THREADS=3
```

2.4.2 N-way BLAS

If using the ClearSpeed n-way BLAS library, the following variables may be used. These can only be set as environment variables.

CS_NWAY_HOST

Set this variable to specify the BLAS library to use on the host. The default value is empty, which means that no host library would be used by n-way. You *must* set this value.

For information on which BLAS libraries are supported, see 1.3.2, *Supported host BLAS libraries*, on page 7.

```
CS_NWAY_HOST=/usr/lib/libacml.so
```

CS_NWAY_DGEMM

Set this variable to specify the number of threads used for DGEMM when running on the host. This is typically set to the number of cores on the host. The default value is 1.

```
CS_NWAY_DGEMM=4
```

CS_NWAY_DTRSM

Set this variable to specify the number of processors or threads used for DTRSM when running on the host. This is typically set to the number of cores on the host. The default value is 1.

```
CS_NWAY_DTRSM=4
```

Example

A typical setup, optimized for LINPACK on a four-core machine, is as follows:

```
CS_BLAS_TRACING=0
CS_BLAS_CONSOLE_TRACING=1
CS_BLAS_TRACE_FILE=/proc/self/fd/1
CS_HOST_THREADS=1
CS_BLAS_HOST_ASSIST_PERCENTAGE=25
CS_NUMBER_OF_ASSIST_THREADS=3
CS_NWAY_DGEMM=4
CS_NWAY_DTRSM=4
CS_HOST_BLAS=/lib/libnWayBLAS.so:/lib/libgoto_opt-64_1024-r0.97.so
CS_NWAY_HOST=/lib/libgoto_opt-64_1024-r0.97.so
```

3 Using CSXL with MATLAB

The following sections describes how to use CSXL with MATLAB.

CSXL accelerates the Level 3 BLAS dgemm function for doing double-precision matrix-matrix multiply. CSXL intercepts calls to the host BLAS library and, if the matrix sizes are suitably large, accelerates the processing on the *Advance* board.

Environment variables are used to define the location of the CSXL library and the host BLAS library. These need to be set up to allow MATLAB to use the CSXL library to accelerate calls to the supported BLAS functions. These variables are described in section 2.4, *Configuration variables*, on page 9.

3.1 Microsoft Windows

The following describes how to use CSXL with MATLAB on a Windows XP operating system.

CSXL accelerates the Level 3 BLAS dgemm function for doing double-precision matrix-matrix multiply. CSXL intercepts calls to the host BLAS library and, if the matrix sizes are suitably large, offloads the processing to the *Advance* board.

Before you start, ensure the following:

- MATLAB is installed at `c:\program files\matlab\RVER` where `RVER` is either r14sp2 or r2006a
- ClearSpeed runtime and diagnostics are in path: `c:\program files\clearspeed\csx600_m512_le`

To run MATLAB with ClearSpeed acceleration:

1. Open a DOS shell window and go to the directory where you want to run MATLAB.
2. Ensure the following ClearSpeed environment variables have been set either by typing them on the command line or by creating a batch file:
   ```
   set CS_HOST_BLAS=c:\program files\matlab\RVER\bin\win32\mkl.dll
   set BLAS_VERSION=c:\program files\clearspeed\csx600_m512_le\bin\blas_cs.dll
   ```
3. Run MATLAB, for example:
   ```
   matlab -nojvm -r csxdgemm
   ```
   where `csxdgemm.m` is an M-file in the local directory.

3.2 Linux

The following describes how to use CSXL with MATLAB on a Linux operating system.

```
export BLAS_VERSION=/opt/clearspeed/csx600_m512_le/lib/libblas_cs.so
export CS_HOST_BLAS=/usr/local/matlabR14/bin/glnx86/libmkl.so
export LD_LIBRARY_PATH=/usr/local/matlabR14/bin:$LD_LIBRARY_PATH
```

The values of these variables will vary depending on the version of MATLAB, which host BLAS library is being used, etc.

An example bash shell script to initialize the environment, called `cs_setup_matlab`, is included below.

Before starting MATLAB, use the bash shell script to initialize the environment. For example:

```
source cs_setup_matlab
matlab
```

3.2.1 Example environment set-up script

The following shell script, `cs_setup_matlab`, can be used to initialize the environment for running MATLAB with the *Advance* board and the CSXL library.

```
echo "MATLAB setup environment for Advance board in single-processor system"
echo ""
echo "--------------------------------------------------------------------"

# Set up the basic ClearSpeed environment variables
# -------------------------------------------------
export CS_HOME=/opt/clearspeed/csx600_m512_le
echo "source $CS_HOME/bin/bashrc"
source $CS_HOME/bin/bashrc

# Point to the MATLAB installation
# --------------------------------
export MATLAB=/usr/local/Matlab

# Have MATLAB report which BLAS library is in use
# -----------------------------------------------
export LAPACK_VERBOSITY=1

# Point to the ClearSpeed BLAS library
# ------------------------------------
export BLAS_VERSION=$CS_HOME/lib/libblas_cs.so

# Choose a host BLAS library
# --------------------------
# MATLAB Intel MKL BLAS 32-bit library
export CS_HOST_BLAS=$MATLAB/bin/glnx86/libmkl.so

# Intel MKL BLAS 32-bit library
#export CS_HOST_BLAS=/opt/intel/mkl721/lib/32/libmkl.so

# Add CSXL to the dynamic library path
# ----------------------------
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CS_HOME/lib:$MATLAB/bin/glnx86:/opt/intel/
mkl721/lib/32

# Echo the ClearSpeed environment to the terminal
# -----------------------------------------------
env | grep BLAS_VERSION
env | grep CS

# reset the Advance board
csreset -Av

echo ""
echo "Done "
```

4 Using CSXL with Mathematica 5.2

The following describes how to use CSXL with Mathematica on a Linux operating system.

Mathematica 5.2 uses Intel's Math Kernel Library (MKL) as its default standard math library. MKL is a multi-component library with separate shared object files for BLAS, LAPACK, and other sets of math functions.

4.1 Step 1: Create a symbolic link to intercept MKL

To allow Mathematica to use the CSXL library for its BLAS calls:

1. Change to the directory:

   ```
   cd /opt/clearspeed/csx600_m512_le/lib
   ```

2. Create a symbolic link:

   ```
   ln -s libblas_cs.so libmkl_lapack32.so
   ```

4.2 Step 2: Configure Mathematica to set up the CSXL environment

In the following instructions, the path to the Mathematica installation is referred to as `$MATHEMATICA`.

To enable access to the CSXL library from within Mathematica, perform the following steps once:

1. Log in as the person who installed Mathematica. That is, Mathematica can only be configured by the person who installed it.
2. Change to the `$MATHEMATICA/Executables` directory:

   ```
   cd $MATHEMATICA/Executables
   ```

3. Copy the file `math` to a new name, for example `math_csx`:

   ```
   cp math math_csx
   ```

4. Open the new file in a text editor and find the line:

   ```
   exec "${MathKernel}" "$@"
   ```

5. Modify the script to load the CSXL library before starting Mathematica by adding the following lines before the `exec` line (this example is specific to SLES 9.3):

   ```
   export CS_BLAS_TRACING=0
   export CS_BLAS_TRACE_FILE=/proc/self/fd/1
   export CS_BLAS_CONSOLE_TRACING=1
   __PFIX__=/opt/mathematica/SystemFiles/Libraries/Linux-x86-64
   export LD_LIBRARY_PATH=/opt/clearspeed/csx600_m512_le/lib/:$LD_LIBRARY_PATH
   export CS_HOST_BLAS=$__PFIX__/libmkl_lapack32.so:
                      ↳$__PFIX__/libmkl_lapack64.so:$__PFIX__/libmkl.so:
                      ↳$__PFIX__/libmkl_vml_p4n.so
   export CS_HOST_THREADS=1
   exec "${MathKernel}" "$@"
   ```

   Where the symbol ↳ indicates that a long line has been broken to fit the page. This should be typed as a single line.

   Note: For 32-bit Linux systems, remove `-x86-64` from the `__PFIX__` definition.

6. Use the shell script provided to set up the environment variables:
   ```
   source /opt/clearspeed/csx600_m512_le/bin/bashrc
   ```
7. Reset the board:
   ```
   csreset -Av
   ```

4.3 Step 3: Configure a Mathematica kernel to use CSXL

To configure the Mathematica kernel:

1. Start the Mathematica front end in the normal way:
   ```
   mathematica
   ```
2. Create a kernel entry for the modified `math_csx` script. From the **Kernel** menu, select **Kernel Configuration Options..**
3. Click the **Add...** button of the "Properties" dialog box.
4. Enter a value for **Kernel Name**, such as "ClearSpeed".
5. Enter "`math_csx`" in the **Shell Command to Launch Kernel** field.
6. Click **OK**.
7. Click **OK** in the "Properties" dialog.

4.4 Step 4: Select the accelerated kernel for a notebook

Once a notebook has been opened or created in Mathematica, the CSXL-accelerated kernel can be selected from the **Kernel** menu as follows:

1. Select the kernel from the **Notebook's Kernel** menu item.
2. Start the kernel from the **Start Kernel** menu item.
3. Evaluate the notebook by selecting **Evaluate Notebook** from the **Evaluation** menu item.

5 Functions implemented

5.1 BLAS

For information on compatible compilers, see 1.3.3, *Supported compilers*, on page 7.

5.1.1 DGEMM

Operation

DGEMM performs one of the matrix-matrix operations

```
C := alpha*op(A)*op(B) + beta*C,
```

where `op(X)` is one of:

- `op(X) = X`
- `op(X) = X'`

`alpha` and `beta` are scalars, and `A`, `B` and `C` are matrices, with `op(A)` an `m` by `k` matrix, `op(B)` a `k` by `n` matrix and `C` an `m` by `n` matrix.

C prototype

```
extern void dgemm_(char *transa, char *transb,
                    int *m, int *n, int *k,
                    double *alpha, double *a,
                    int *lda, double *b, int *ldb,
                    double *beta, double *c,
                    int *ldc, int transa_len, int transb_len);
```

FORTAN interface

```
      SUBROUTINE DGEMM ( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB,
     $                   BETA, C, LDC )
*     .. Scalar Arguments ..
      CHARACTER*1        TRANSA, TRANSB
      INTEGER            M, N, K, LDA, LDB, LDC
      DOUBLE PRECISION   ALPHA, BETA
*     .. Array Arguments ..
      DOUBLE PRECISION   A( LDA, * ), B( LDB, * ), C( LDC, * )
```

Parameters

TRANSA - CHARACTER*1

- On entry, TRANSA specifies the form of op(A) to be used in the matrix multiplication as follows:
  - TRANSA = 'N' or 'n', op(A) = A.
  - TRANSA = 'T' or 't', op(A) = A'.
  - TRANSA = 'C' or 'c', op(A) = A'.
- Unchanged on exit.

TRANSB - CHARACTER*1

- On entry, TRANSB specifies the form of op(B) to be used in the matrix multiplication as follows:
  - TRANSB = 'N' or 'n', op(B) = B.
  - TRANSB = 'T' or 't', op(B) = B'.
  - TRANSB = 'C' or 'c', op(B) = B'.
- Unchanged on exit.

M - INTEGER

- On entry, M specifies the number of rows of the matrix op(A) and of the matrix C. M must be at least zero.
- Unchanged on exit.

N - INTEGER

- On entry, N specifies the number of columns of the matrix op(B) and the number of columns of the matrix C. N must be at least zero.
- Unchanged on exit.

K - INTEGER

- On entry, K specifies the number of columns of the matrix op(A) and the number of rows of the matrix op(B). K must be at least zero.
- Unchanged on exit.

ALPHA - DOUBLE PRECISION

- On entry, ALPHA specifies the scalar alpha.
- Unchanged on exit.

A - DOUBLE PRECISION array of DIMENSION (LDA, ka), where ka is k when TRANSA = 'N' or 'n', and is m otherwise.

- Before entry with TRANSA = 'N' or 'n', the leading m by k part of the array A must contain the matrix A, otherwise the leading k by m part of the array A must contain the matrix A.
- Unchanged on exit.

LDA - INTEGER

- On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANSA = 'N' or 'n' then LDA must be at least max(1, m), otherwise LDA must be at least max(1, k).
- Unchanged on exit.
- **B** - DOUBLE PRECISION array of DIMENSION (LDB, kb), where kb is n when TRANSB = 'N' or 'n', and is k otherwise.
- Before entry with TRANSB = 'N' or 'n', the leading k by n part of the array B must contain the matrix B, otherwise the leading n by k part of the array B must contain the matrix B.
- Unchanged on exit.

LDB - INTEGER

- On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANSB = 'N' or 'n' then LDB must be at least max(1, k), otherwise LDB must be at least max(1, n).
- Unchanged on exit.

BETA - DOUBLE PRECISION

- On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.
- Unchanged on exit.

`C` - DOUBLE PRECISION array of DIMENSION (LDC, n)

- Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry.
- On exit, the array C is overwritten by the m by n matrix (alpha*op(A)*op(B) + beta*C).

`LDC` - INTEGER

- On entry, LDC specifies the first dimension of C as declared in the calling (sub)program. LDC must be at least max(1, m).
- Unchanged on exit.

6 Optimizing use of CSXL

6.1 Performance graphs

The following performance graphs (Figures 6.1 to 6.6) were generated using a system with four dual-core AMD Opteron (Model 870, 2.0 GHz) processors. One core will run DGEMM at about 3.6 GFLOPS. The performance graphs were generated using this machine with one *Advance* accelerator board.

Figure 6.1 shows a typical performance graph for multiplying square matrices (m=n=k). The effect of using different number of cores is clearly shown here. Both CSXL and n-way BLAS were used. Running CSXL/n-way BLAS with two cores used a `CS_BLAS_HOST_ASSIST_PERCENTAGE` of `4`. Four cores used a host-assist percentage of 19 and eight cores used a host assist percentage of 40.

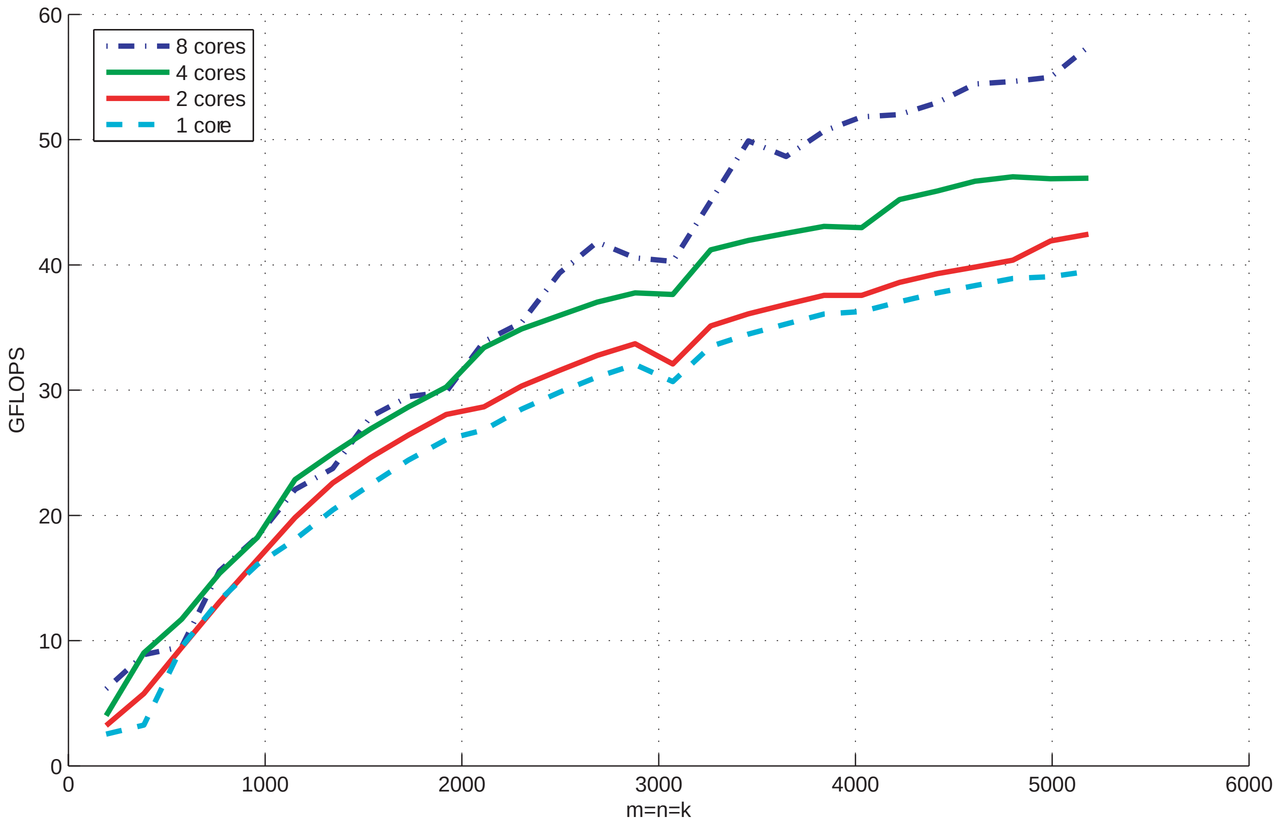

Figure 6.1 CSX600 square DGEMM performance

Figure 6.2 shows the performance for DGEMM sizes which typically take a lot of time when running Linpack (LU decomposition). Here, m=n and k is much smaller than m (see also Figure 6.3). This time the number of cores used is fixed to two (`CS_NUMBER_OF_ASSIST_THREADS` is set to 1).

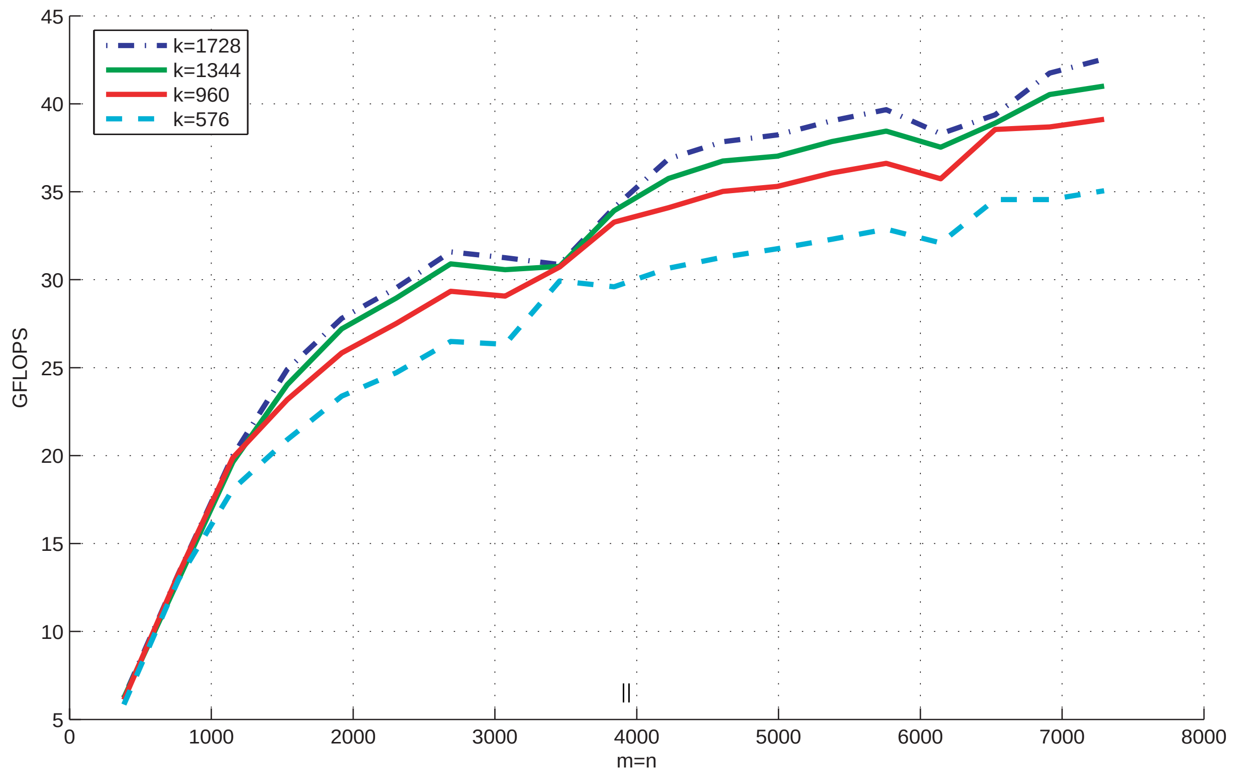

Figure 6.2 CSX600 LU-shape DGEMM performance

Figure 6.3 shows A is high and narrow, B is wide and short, C is large and square.

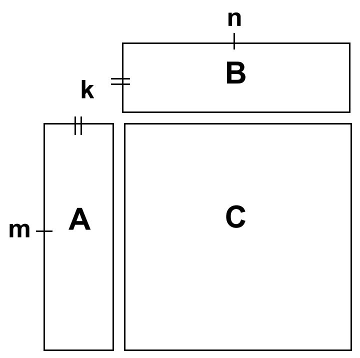

Figure 6.3 Illustration of CSX600 LU-shape DGEMM

In Figure 6.4, m=k and n is much smaller than m . The total number of cores used is fixed at one.

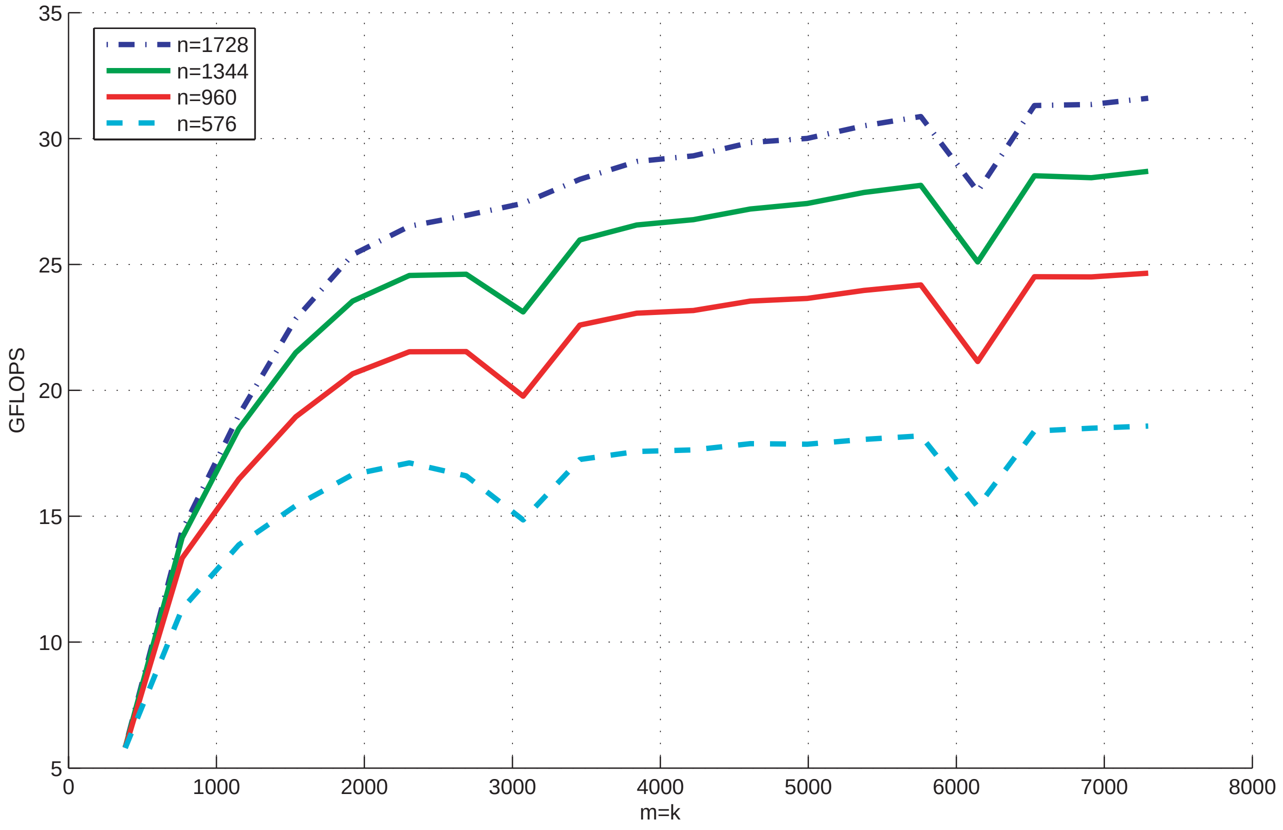

Figure 6.4 CSX600 square matrix A DGEMM performance

Figure 6.5 shows A is large and square, B is high and narrow, C is high and narrow.

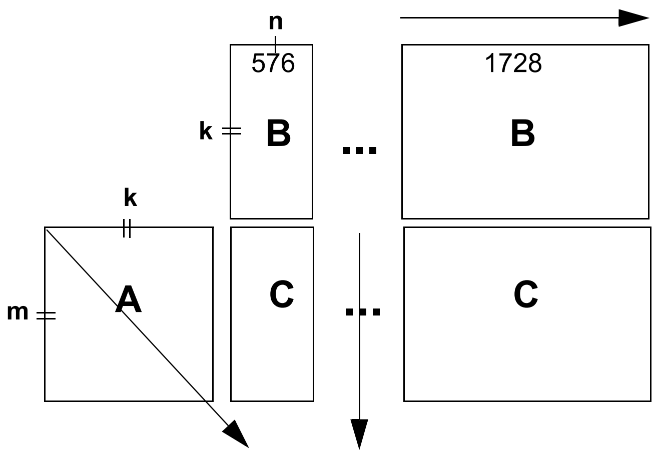

Figure 6.5 Illustration of CSX600 square matrix A DGEMM

In Figure 6.6, k=n and m is much larger than k. The total number of cores used is fixed at one.

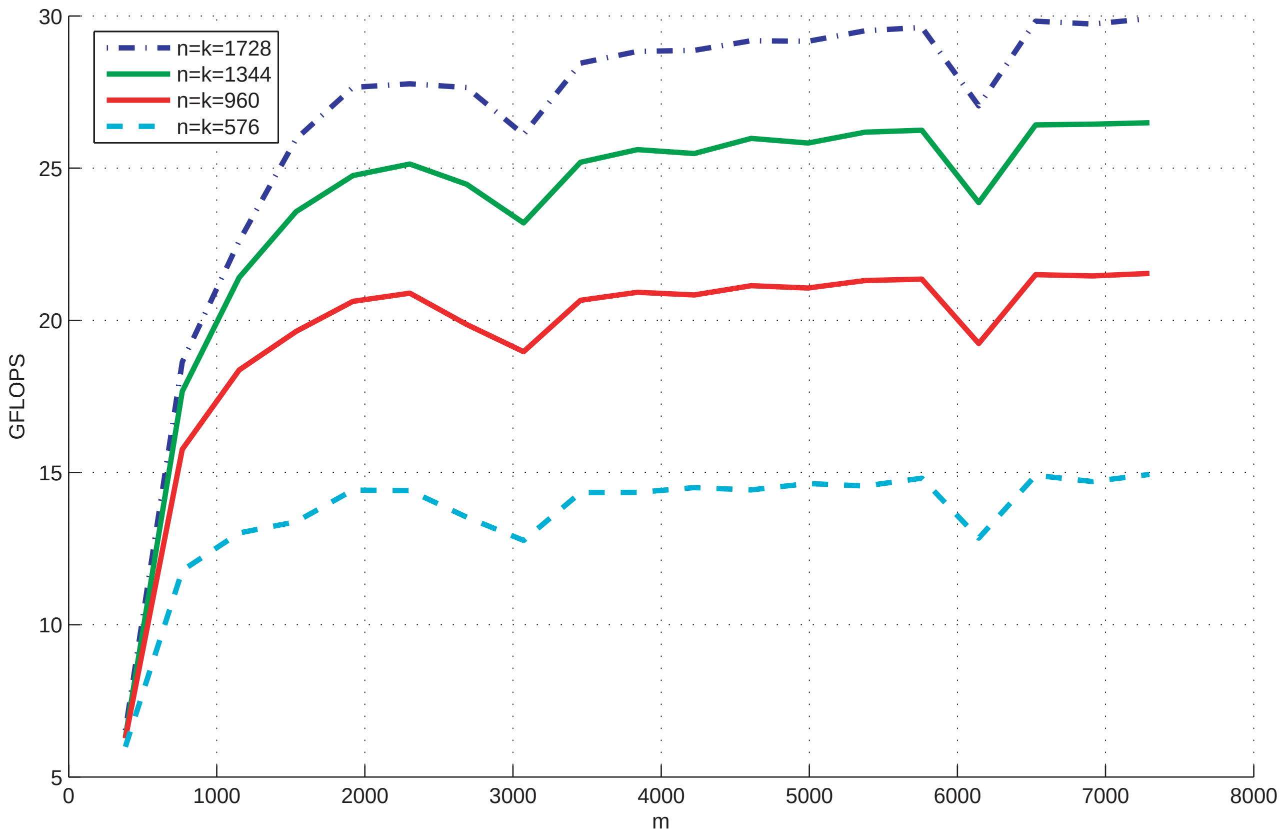

Figure 6.6 CSX600 square matrix B DGEMM performance

Figure 6.7 shows A is high and narrow, B is small and square, C is high and narrow.

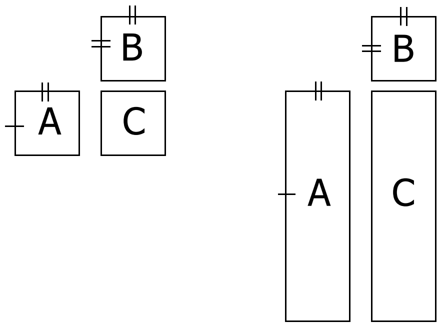

Figure 6.7 Illustration of CSX600 square matrix B DGEMM

6.2 Calculating host fraction on Linux

This section describes how to aggregate the performance of the host and the *Advance* accelerator board by partitioning the workload between them. The following instructions take you through the steps needed to calculate the host fraction (LINPACK is used in this example):

1. Compile the following program (this is provided as `dgemmtest.c` in the `examples/csxl` directory of the installation):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

extern void dgemm_(char *transa, char *transb, int *m, int *n, int *k,
                  double *alpha, double *a,
                   int *lda, double *b, int *ldb, double *beta, double *c,
                   int *ldc, int transa_len, int transb_len);

int main(int argc, char *argv[])
{
  double *a, *b, *c, alpha, beta;
  size_t m,n,k;
  struct timeval start, end;

  if (argc<4)
    {
      printf("Usage: %s m n k to call a dgemm with size m,n,k\n", argv[0]);
      return 0;
    }

  m=atoi(argv[1]);
  n=atoi(argv[2]);
  k=atoi(argv[3]);

  a=(double*)malloc(m*k*sizeof(double));
  b=(double*)malloc(n*k*sizeof(double));
  c=(double*)malloc(n*m*sizeof(double));

  memset(a, 0, m*k*sizeof(double));
  memset(b, 0, n*k*sizeof(double));
  memset(c, 0, m*n*sizeof(double));

  alpha=2.75; beta=4.21;

  dgemm_("T", "T", &m, &n, &k, &alpha, a, &k, b, &n, &beta, c, &m, 1, 1);

  free (c);
  free (b);
  free (a);

  return 0;
}
```

2. Make sure you have the CSXL package installed and then compile the code as follows:

```
$> gcc -lm -lg2c -lpthread dgemmtest.c /opt/clearspeed/csx600_m512_le/lib/libblas_cs.so
```

3. Setup the CSX runtime and CSXL packages:

```
$> source /opt/clearspeed/csx600_m512_le/bin/bashrc
$> csreset -Av
$> export CS_BLAS_TRACING=1
$> export CS_BLAS_TRACE_FILE=`tty`
$> export CS_BLAS_CONSOLE_TRACING=1
$> export CS_HOST_BLAS=/opt/clearspeed/csx600_m512_le/lib/libnWayBLAS.so
```

4. Specify the number of cores you have in the machine. For example, the following assumes you have 4 cores. If you have a different number, replace 3 (four minus one) below, with the number of cores you have less one:

```
$> export CS_NWAY_DGEMM=3
```

5. Specify the name of the native host BLAS you are using. For example, the following uses ACML. If you use something other than ACML, replace `libacml.so` with the name of the host BLAS you have selected:

```
$> export CS_NWAY_HOST=/usr/lib/libacml.so
```

6. Take the following two measurements:

   a. Firstly, measure the capability of the host (*hosttime)*:

```
$> export CS_BLAS_HOST_ASSIST_PERCENTAGE=100
$> ./a.out 15360 15360 960
```

A description of the trace data that is displayed can be found in the description of CS_BLAS_TRACING, on page 9.

The last line will show the time in microseconds. For example:

```
... _t=35062405]
```

   b. Next, measure the capability of the installed card (*cardtime)*:

```
$> export CS_BLAS_HOST_ASSIST_PERCENTAGE=0
$> ./a.out 15360 15360 960
```

The last line will show the time in microseconds. For example:

```
... _t=11226450]
```

7. Use the above figures (`t=35062405` and `t=11226450)` to derive the first approximation of the host fraction for LINPACK Q=P=1:

cardtime/(*cardtime*+*hosttime*)
11226450/462888555 or 24%

Note: The above timings were made with a 4-core host. However, timings and host assist may vary.

7 Tuning LINPACK

This chapter gives specific instructions on how to adjust the LINPACK for the *Advance* accelerator board.

7.1 Getting started

It is assumed that the software packages shown in Table 7.1 are preinstalled and running on the system before tuning is performed.

Package	**Description**	**Source**
csx600-m512_le-runtime-2.1-1.64.1.22[a]	Driver package for ClearSpeed *Advance* accelerator board.	http://support.clearspeed.com/downloads/
csx600-m512_le-csxl-2.1-1.29.1.24[a]	Math library package for ClearSpeed *Advance* accelerator board.	http://support.clearspeed.com/downloads/
ACML	AMD Core Math Library (includes BLAS).	http://developer.amd.com/acml.aspx
HPL	High-performance LINPACK benchmark.	http://www.netlib.org/benchmark/hpl
GotoBLAS	Vendor neutral BLAS.	http://www.tacc.utexas.edu/resources/software
MPICH	Generic MPI library for building HPL.	http://www-unix.mcs.anl.gov/mpi/mpich1

Table 7.1 Software packages used

a. Please note that version numbers may vary.

It is also assumed that the Linux platform is used and that you have access to an optimized MPI implementation for your interconnect hardware.

To configure and run LINPACK:

1. Build HPL with the optimized MPI, and use CSXL (`libblas_cs.so`) as `$LAlib` in the architecture Makefile with the following line:

   ```
   LAlib = -L$(LAdir) -lblas_cs -lg2c -lm -ldl -lpthread
   ```

2. Set the following environment variables for n-way BLAS:

   ```
   CS_NWAY_HOST,
   CS_NWAY_DGEMM
   CS_NWAY_DTRSM
   ```

   Set these variables to the values described in 2.4.2, *N-way BLAS*, on page 10, for example:

   ```
   export CS_NWAY_DGEMM=8
   export CS_NWAY_DTRSM=8
   export CS_NWAY_HOST=/usr/lib/libacml.so
   ```

3. Set the environment variables for CSXL by using `$CSXL_CONFIG_FILE` to point to a text file containing the variables in 2.4.2, *N-way BLAS*, on page 10:

   ```
   export CSXL_CONFIG_FILE=$PWD/CSXL.cfg
   cat $CSXL_CONFIG_FILE
   ```

   ```
   CS_HOST_BLAS=/opt/clearspeed/csx600_m512_le/lib/libnWayBLAS.so:/usr/lib/libacml.so
   CS_BLAS_HOST_ASSIST_PERCENTAGE=37
   CS_HOST_THREADS=1
   CS_NUMBER_OF_ASSIST_THREADS=7
   ```
4. Reset the board with `csreset`.

   Note: The board must be reset when you first run LINPACK. It is not necessary to reset it each time you run LINPACK.
5. Modify the HPL parameter file (`HPL.dat`) to run a sizeable job with blocking factors that are more suitable than the default. See the output log on page 31 for a full example of a parameter set, with the most significant parameters being:

   N ranging between 9600 for a small test and 56640 for a very large test.

   NB = 960 for a more suitable blocking factor.
6. Run HPL (`xhpl`).

7.2 Predicting performance on a single node

Blocked LU factorization performed in the LINPACK benchmark can be tuned to exploit efficient underlying library routines including matrix multiply (see also DGETRF in [1], [2], [3]).

The number of floating point operations (FLOPs) to solve a system of N unknowns is 2/3 x $N^3 + 2N^2$. Using a block-based implementation, the major computation in terms of runtime or FLOP count is the update step executed with

DGEMM. Figure 7.1 shows the percentage of total FLOPs spent in these update computations with various block sizes. For even very large block sizes, the amount of time spent in these calls is large.

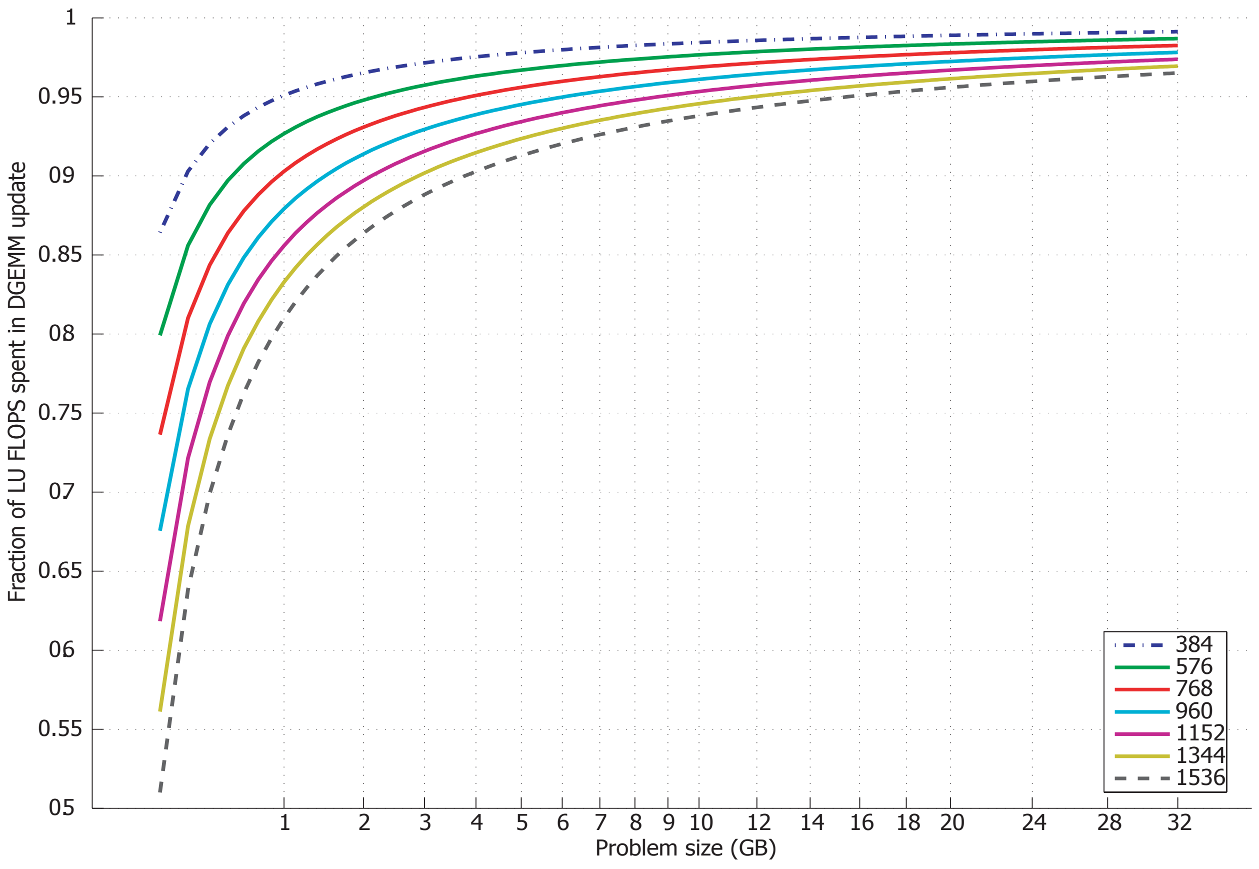

Figure 7.1 Fraction of FLOPs spent in DGEMM update step for LU factorization

The x-axis in Figure 7.1 represents the problem size in the total memory requirements. The x-axis is linear in N, the number of unknowns, and square in the memory requirements. The problem size is $8N^2$ bytes as the computation is performed with 64-bit precision arithmetic.

Since these DGEMM steps are so dominant, you can use the values in Figure 7.1 as a first approximation to LINPACK performance along with the corresponding DGEMM performance of the machine. ClearSpeed's coprocessors assist the host with computationally intensive functions such as DGEMM. Since the resulting performance will be an aggregate of the host and the coprocessor, you construct the following model for estimating LINPACK performance:

$$(H(C-1)/C + B)f \qquad \textit{Equation 7.1}$$

Where:

f is the fraction of time spent in DGEMM. For example, Figure 7.1 shows all blocking factors use over 90% when the problem size is over 8 GB.

H is the performance of DGEMM on the host. For example, a four-way Opteron 870 with eight cores and 32 GB of memory can achieve 28.89 GFLOPS with DGEMM.

C is the number of cores in the host machine. For example, this machine has eight.

B is the performance of DGEMM on the *Advance* accelerator board.

The DGEMM performance of the *Advance* accelerator board alone is depicted in Figure 7.2. The x-axis in Figure 7.2 represents the M dimension of the input and output matrices for this particular DGEMM operation. The other two matrix dimensions (N and K) are defined as $N=M+1$ and K is the blocking factor as shown in Figure 7.1. The relationship between M and N is a function of the LINPACK parameter set, specifically the $P = Q = 1$ values. Again, the x-axis is linear in the number of unknowns in the system as per Figure 7.1 with the result of the DGEMM being square in memory size as per Figure 7.1. There is, therefore a semantic link between the two figures to exploit in the estimation of performance in the context of a coprocessor accelerated DGEMM.

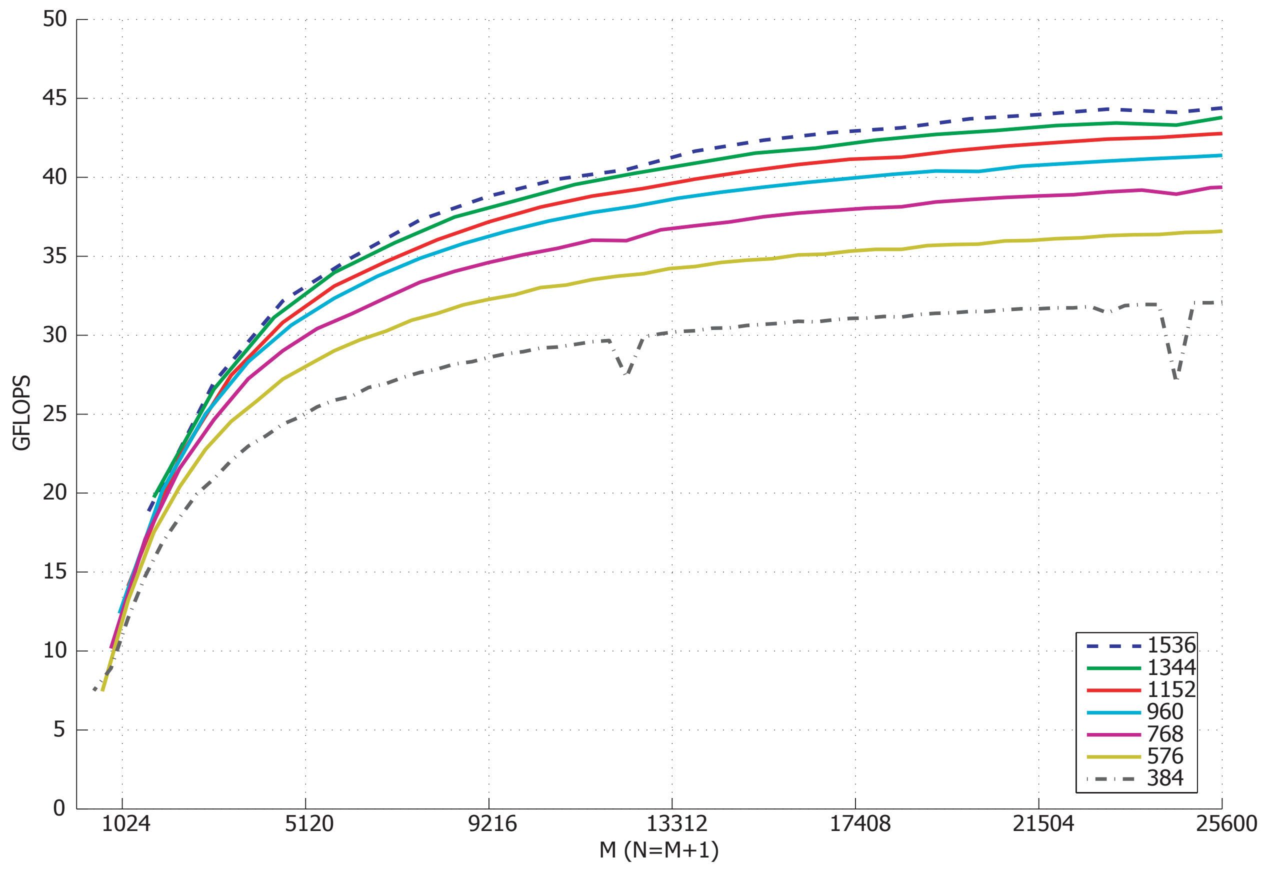

Figure 7.2 DGEMM performance of coprocessors

Returning to the example Opteron system with a (host) capability of *H* =28.89 GFLOPS of DGEMM, make an estimate of performance of the aggregated system:

(H(C-1)/C + B)f *Equation 7.2*

Where: H=28.89, C=8, B=42.69 and f=96% (for block-size 960 and a large problem) the result is 65.25 GFLOPS. Compare this to the 27.03 GFLOPS LINPACK score that the machine achieves without an *Advance* accelerator board*.*

Choosing the optimal block size is a function of the individually achievable DGEMM score and the amount of time spent in these DGEMMs. For example, at a block size of 384 and a problem of 56,640 unknowns, 147 update DGEMMs are run and most will be at the upper performance level of ~33 GFLOPS on the board plus the host contribution. Compare this to the largest example block size of 1536 that will execute only 36 update DGEMMs where only a handful will reach the ~46 GFLOPS on the board plus the host contribution. Clearly the larger blocking factors are favorable generally, but the 56,640 unknowns consumes 24 Gb of memory and it is difficult to see how a large enough problem could easily be deployed on a single node to properly exploit the higher asymptotic DGEMM score.

Decomposing the estimate, start with the host. With eight cores, each core is contributing 3.61 GFLOPS towards the DGEMM score. Since you require at least one core to perform data management operations to service the coprocessor board, you can estimate the host contribution to be 25.28 GFLOPS of DGEMM. Selecting a blocking factor NB=960, you will see that the coprocessors achieve 42.69 GFLOPS. The sum of 67.97 GFLOPS is then multiplied by the fraction of computation that they represent in a LINPACK computation. Choosing a problem size of 24 Gb (56,640 unknowns), you see approximately 96% of the FLOPs are executed with these DGEMM updates. The first approximation of LINPACK performance for this point would be 65 GFLOPS. This will be an optimistic estimate, as only the very largest update DGEMMs will achieve the high GFLOPS scores. Figure 7.2 shows that smaller update DGEMMs will achieve lower DGEMM scores as LINPACK progresses.

Figure 7.3 shows the performance curve for LINPACK on the reference machine. From it you can make at least two observations. Firstly, you can measure the rate at which the performance approaches the estimated performance and observe how accurate the estimate is. A second function of this graph correlates problem size with performance. Should you have a specific problem, you may read off the expected performance.

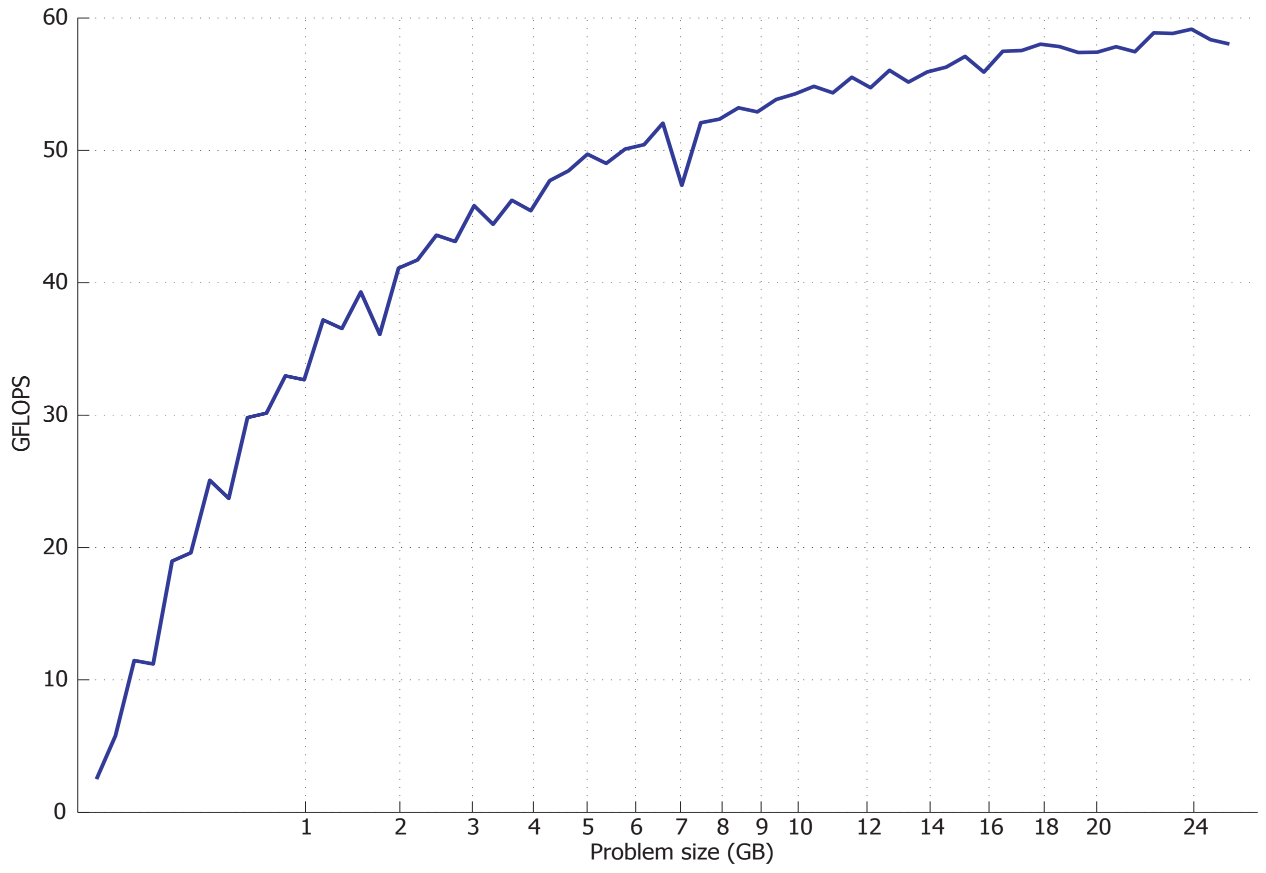

Figure 7.3 LINPACK performance of Opteron system with ClearSpeed

The recorded LINPACK performance for this system at 56,640 unknowns with a blocking factor of 960 is 59.46 GFLOPS. This is within 10% of the admittedly optimistic estimate of 65 GFLOPS above. The following shows the output for running LINPACK:

```
HPLinpack 1.0a  --  High-Performance Linpack benchmark  --   January 20, 2004
Written by A. Petitet and R. Clint Whaley,  Innovative Computing Labs.,  UTK

An explanation of the input/output parameters follows:
T/V    : Wall time / encoded variant.
N      : The order of the coefficient matrix A.
NB     : The partitioning blocking factor.
P      : The number of process rows.
Q      : The number of process columns.
Time   : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:

N      :   56640
NB     :     960
PMAP   : Row-major process mapping
P      :       1
Q      :       1
PFACT  :    Left
NBMIN  :       8
NDIV   :       2
RFACT  :   Crout
BCAST  :   1ring
DEPTH  :       0
SWAP   : Mix (threshold = 960)
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words

T/V                N    NB     P     Q               Time             Gflops
WR00L2C8       56640   960     1     1            2037.49          5.946e+01
||Ax-b||_oo / ( eps * ||A||_1  * N        ) =        0.0037512 ...... PASSED
||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  ) =        0.0046283 ...... PASSED
||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo ) =        0.0008922 ...... PASSED
```

7.3 Understanding the system configuration

The configuration detail is important in understanding the mechanics that combined the Opteron machine with the CSX600 coprocessors.

The ClearSpeed accelerated math library (CSXL) package is placed between the end application and the current vendor BLAS library (termed *host-BLAS*, for example ACML, Goto-BLAS and so on). There are two component libraries in the CSXL package. The first is the CSXL library. This provides the infrastructure and algorithms to decide which BLAS routines involve CSX600 acceleration and use of the host possibly via n-way BLAS. The second is n-way BLAS. This is a thin software layer using pthreads to provide explicit, dynamic and lightweight control over the number of threads on the host on a call-by-call basis. Then, for example, when the CSX600 processors are selected for inclusion in a BLAS computation, a different number of host cores may be used to when the CSX600 processors are not. LINPACK has the scenario shown in Figure 7.4, where the arrows indicate a run-time dependency for BLAS (for example, DGEMM).

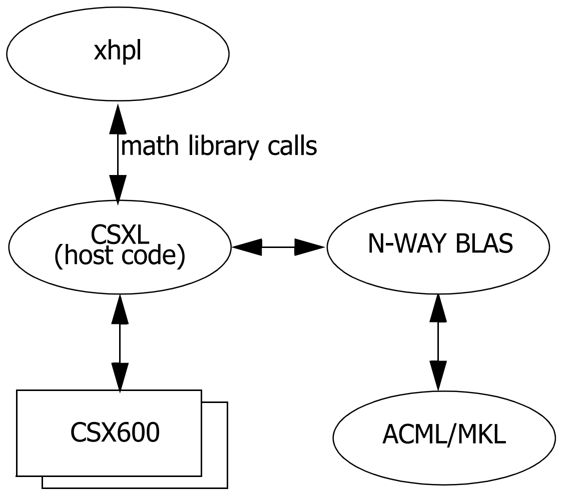

Figure 7.4 xhpl with CSXL

The dependency between the application `xhpl` (LINPACK) and CSXL is the build-time dependency. The other dependencies are created through a set of configuration variables.

`CS_HOST_BLAS` specifies a list of BLAS libraries which implement the full set of BLAS and LAPACK functions. See page 9 for more details. This is the link pair between CSXL and n-way BLAS and host BLAS.

Similarly, `CS_NWAY_HOST` specifies the BLAS library to use on the host. See page 10 for more details. This is the link between n-way BLAS and host BLAS.

Most configuration parameters represent simple physical properties of the machine, such as the number of cores used for component functions. Others are empirically based and require some tuning to find the best performance as described in 2.4, *Configuration variables*, on page 9.

Briefly, the parameters include:

For CSXL:

- `CSXL_CONFIG_FILE`

  A file to collect the CSXL parameter variables and not interfere with the general environment.
- `CS_HOST_BLAS`

  A colon separated list of vendor BLAS. For example ACML, Goto-BLAS, MKL, ATLAS and so on.
- `CS_BLAS_HOST_ASSIST_PERCENTAGE`

  The fraction of large DGEMMs that is executed on the host in parallel to execution on the *Advance* accelerator board to aggregate performance.
- `CS_HOST_THREADS`

  The number of threads to perform data management functions. Typically set to a value of one (1) for LINPACK experiments.
- `CS_NUMBER_OF_ASSIST_THREADS`

  The number of threads involved in computing the host fraction of large DGEMMs. Typically one less than the number of host cores.

For n-way BLAS:

- `CS_NWAY_HOST`
  A vendor BLAS as for CSXL's `CS_HOST_BLAS`.
- `CS_NWAY_DGEMM`
  The number of host threads used to compute DGEMM. Typically the number of host cores. Specifically, this value is that used when the *Advance* accelerator board is not involved in a computation.
- `CS_NWAY_DTRSM`
  The number of host threads used to compute DTRSM. Typically the number of host cores.

Table 7.2 lists the parameters used in the experiments presented above.

Parameter	**Value**
xhpl (LINPACK) BLAS	CSXL BLAS (`libblas_cs.so`)
CSXL host BLAS (`CS_HOST_BLAS`)	n-way BLAS & host-BLAS (`libNWayBLAS.so:libacml.so`)
Host fraction (`CS_BLAS_HOST_ASSIST_PERCENTAGE`)	37
Number of host cores used DGEMM with ClearSpeed (`CS_HOST_THREADS`) (`CS_NUMBER_OF_ASSIST_THREADS`) DGEMM without ClearSpeed (`CS_NWAY_DGEMM`)	 1 7 8
Additional N-way BLAS parameters (`CS_NWAY_HOST`) (`CS_NWAY_DTRSM`)	 GOTO-BLAS (0.97) or ACML 3.0 (`libacml.so`) 8

Table 7.2 Acceleration parameters

Determining the correct host fraction (`CS_BLAS_HOST_ASSIST_PERCENTAGE`) starts with a first approximation that is then refined through empirical measurements. Use the following first approximation method to define the host-assist fraction (a):

$$a = H((C-1)/C)/(B+H(C-1)/C) \qquad \textit{Equation 7.3}$$

Where: as before, H is the host DGEMM capability, C is the number of host cores running DGEMM and B is the board DGEMM capability. Assume that one host core is managing data between the host memory and the *Advance* accelerator board.

Since the tuning is specifically for known DGEMM shapes, trivial variation in the first approximation value will find the maximum. For this system, there is a host capability after scaling ((C-1)/C) of 25.28 GFLOPS and a coprocessor capability of 42.69 GFLOPS. The fraction of the sum of these two parts on the part of the host is 37% that is (fortunately) the same as the empirically found best. To show the compounded effect of a nonoptimal host fraction on LINPACK performance, Figure 7.5 shows the difference between the optimal and a slightly higher fraction.

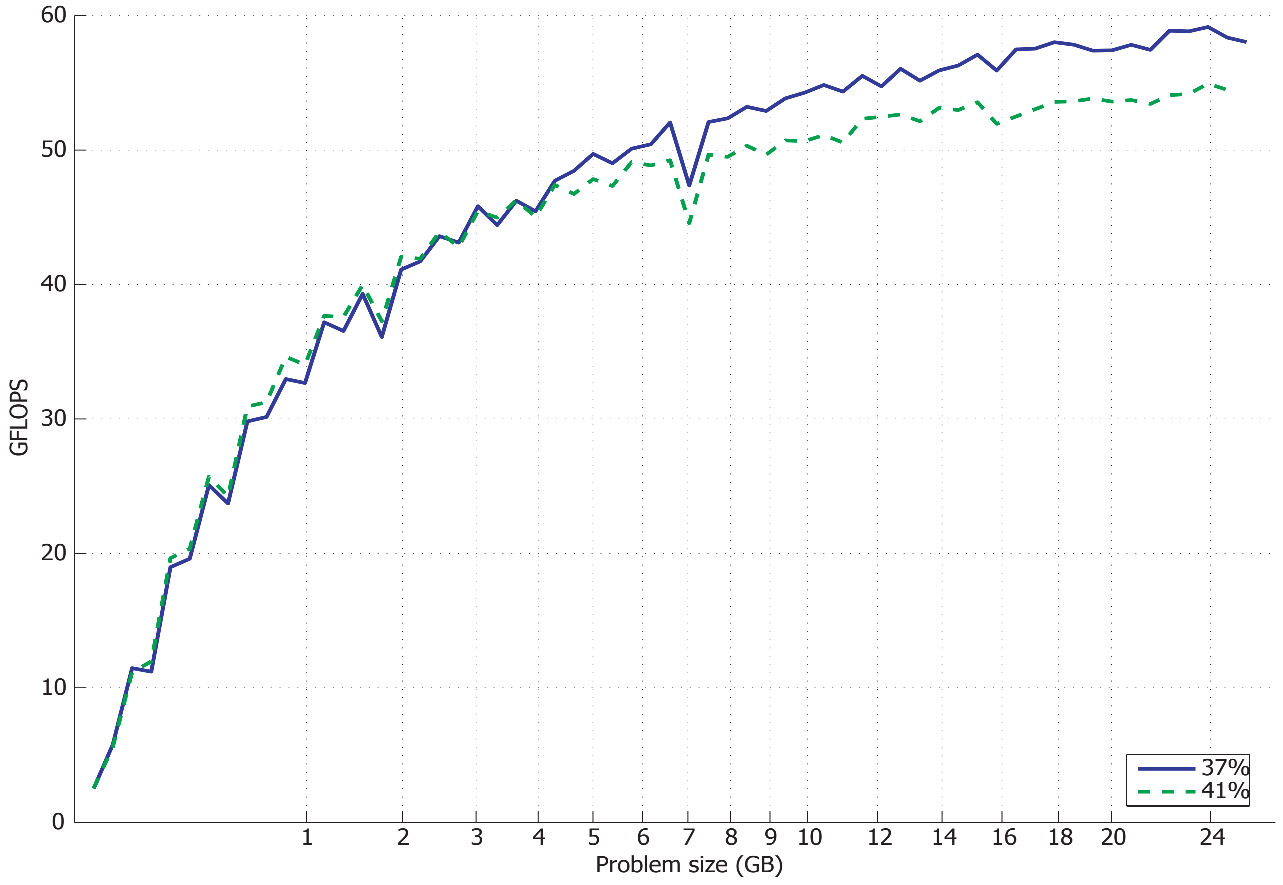

Figure 7.5 Host fraction suboptimality effect on LINPACK performance

7.4 Conclusions for single-node LINPACK

Estimating, configuring and then tuning the performance of ClearSpeed augmented systems for LINPACK is a straightforward process for host systems with moderate numbers of cores. This chapter describes the process with an eight-core system and records more than a 100% increase in LINPACK performance in the aggregated system.

For numerical computations that have significant computational steps, such as QR and Cholesky factorization, the process should be similar, if not identical with specific metrics for the factorization selected.